Explorations of the BDI Multi-Agent support for the Knowledge Discovery in Databases Process

Alejandro Guerra-Hernández, Rosibelda Mondragón-Becerra, and Nicandro Cruz-Ramírez
Departamento de Inteligencia Artificial
Universidad Veracruzana
Facultad de Física e Inteligencia Artificial
Sebastián Camacho No. 5, Xalapa, Ver., México, 91000
{aguerra, rmondragon, ncruz}@uv.mx

Abstract. Knowledge Discovery in Databases (KDD) is the process of finding valid, novel, useful and understandable patterns in data, to verify hypothesis of the user or to describe/predict the future behavior of some event. The KDD process involves diverse techniques provided by tools like the Waikato Environment for Knowledge Analysis (WEKA), but usually without guidance. This work is an exploration of the use of Multi-Agent Systems (MAS) methodologies and tools to provide support in the KDD process while using such tools. The Belief-Desire-Intention (BDI) model of agency provides the right level of abstraction to approach this problem. First, the Prometheus methodology is used to analyse the KDD process in terms of MAS of BDI agents. Then, a MAS of decision trees inducers and Bayesian networks builders, that compete to generate the “best” hypothesis for a given KDD problem, is implemented. The main result of this exploration is a framework where it is possible to implement AgentSpeak(L) agents that use primitive actions of WEKA to form intentions for solving problems in the KDD process. Extensions in terms of the number of agents and their capabilities are easy to implement in this framework.

1 Introduction

The Knowledge Discovery in Databases (KDD) process [13] consists in selecting, preprocessing and transforming a data set obtained from several heterogeneous sources such as databases, plain files, data warehouses, etc., in order to facilitate the application of data mining algorithms that obtain the hidden patterns in this dataset. Subsequently, these patterns are interpreted and evaluated to select those that represent useful and novel knowledge [9].

There are several algorithms to carry out each one of the tasks of this process, implemented in tools as Clementine [34], DBMiner [20], Waikato Environment for Knowledge Analysis (WEKA) [36], among others. The difficulty arises when a neophyte has to choose the suitable algorithms for a given problem, since this decision depends on the nature of data, their representation, the mining task,
among other factors. Therefore, KDD is a complex process, that requires well trained users in a variety of disciplines including machine learning, statistics and domain knowledge. But even in this case, some parts of the KDD process can be simply tedious.

Multi-Agent Systems (MAS) have been proposed as a solution to the problems mentioned above. MAS are composed by agents specialized in data mining, which sometimes are distributed in a network [32, 23, 22, 3]. However, we observe that these approaches, often does not provide the level of abstraction required to reason about the KDD process and their participants. This is due to a weak notion of agency, which focuses on basic properties of agents as autonomy, reactivity, pro-activity and social ability [37]. The strong notion of agency conceptualizes and implements the agents as intentional systems. The Belief-Desire-Intention (BDI) model of agency [15, 33] is the best known of these approaches. In this model the agents perform practical reasoning using plans to form intentions to satisfy their desires. The folk-psychology language used in the model, is argued to offer a more effective communication among the participants in a KDD process in order to analyse it to implement support using BDI agents.

Our explorations for such a support are as follows: In order to design a MAS of rational agents that help the human experts in the KDD process, this one is analysed as performed by a MAS of human experts, following the Prometheus methodology [30]. The Prometheus Design Tool was used to get assistance for the design process and to generate a specification of the MAS easy to be implemented. As a result of the analysis we decided to implement a BDI MAS of six agents: a coordinator that receives requests to mine a given data set; a preprocessing specialist; and agents capable to execute ID3, C4.5, Naive Bayes, and TAN algorithms. These last four agents compete to produce the “best hypothesis” for a given problem, and cooperate with the preprocessing agent if necessary. The MAS was implemented in Jason [4], an interpreter of the BDI agent oriented programming language AgentSpeak(L) [33]. The idea was that these agents were capable of executing actions in WEKA [36], as part of their plans. We chose Jason because it is implemented in Java, as well as WEKA, so that the implementation of the intended agents seemed easier in this way.

Finally, the MAS was tested with different data sets, in order to know if such agent competency has sense. The results show that there is not such a thing as the “best method” for all the data sets, and that the exploration automatized by the MAS could be useful. But the main result is that we obtained a framework where it is possible to easily define new agents in the system and to improve the capabilities of the existent ones, i.e., extending their plan library or their beliefs.

The rest of this paper is organized as follows: Section two presents the antecedents, mainly the KDD process and the BDI agents as defined in AgentSpeak(L); section three introduces the tools and methods that were used to implement the MAS; section four comments some aspects of the design and implementation of the KDD process like a BDI MAS; section five reports the results of our experiments; and finally, section six presents the conclusions and discusses future work.
2 Antecedents

In this section, we describe the KDD process, as well as AgentSpeak(L), an agent oriented programming language under the BDI model of agency.

2.1 The KDD Process

KDD refers the non-trivial process of identifying valid, novel, potentially useful and understandable patterns in data [11, 12]. Figure 1 (adapted from Fayyad et al. [11]) shows that the KDD process has an iterative and interactive nature. Results obtained in the process are enhanced incrementally, possibly reconsidering previous decisions in the process [11, 12]. The steps of the KDD process [20, 11, 12] include:

1. Understanding the domain of the application and a background knowledge, as well as identifying the target of KDD process from the point of view of the user.
2. Selecting a subset of variables or a sample of data to create a set of target data, in which the discoveries will take place.
3. Cleaning and preprocessing the set of target data, e.g., removing noise if needed, dealing with missing data, etc.
4. Reducing and projecting data, through the selection of examples and attributes that are important for the target of the KDD process, or using dimensionality reduction or transformation methods to reduce the number

Fig. 1. Steps in the KDD process
of variables under consideration or to find invariant representations for the data.
5. Selecting a data mining procedure, e.g., classification, regression, summarization, clustering.
6. Selecting some data mining algorithms and techniques to search patterns in the reduced data. The data mining expert decides which models or parameters are most appropriate.
7. Searching for patterns of interest (data mining) in a particular representational form as classification trees, rules, regression, clustering, etc.
8. Interpreting and evaluating the mined patterns, possibly reconsidering some of the steps 1 . . . 7.
9. Consolidating the discovered knowledge through its incorporation within another system or simply documenting and sending it to the interested participants in the process.

2.2 AgentSpeak(L) and BDI agents

BDI agents are intentional systems that continuously perceive their environment and take actions to modify it, based on their mental attitudes: beliefs, desires and intentions [15, 33, 8]. Beliefs represent the informational state of the agent. Desires, or goals, represent states that the agent would like to accomplish or bring about, considering its internal or external stimuli. Intentions represent the deliberative state of the agent, e.g., its commitment to some courses of action to accomplish its desires. These courses of action are built from plans in a plan library, e.g., the procedural knowledge of the agent. An event queue is usually used to process perception.

AgentSpeak(L) [33] is the language that was chosen to implement the BDI agents in this work, because it provides an abstract and elegant framework to program such agents [16]. The syntax and semantics of AgentSpeak(L) have been defined formally by means of a grammar and an operational semantics based on a transition system.

The grammar of AgentSpeak(L) as defined for its interpreter Jason [5], is shown in table 1. As usual, an agent \( ag \) is formed by a set of plans \( ps \) and beliefs \( bs \). Each belief \( b_i \in bs \) is a ground first-order term. Each plan \( p \in ps \) has the form \( trigger \ event : context \leftarrow body \). A trigger event can be any update (addition or deletion) of beliefs (at) or goals (g). The context of a plan is an atom, a negation of an atom or a conjunction of them. A non empty plan body is a sequence of actions (a), goals, or belief updates. \( \top \) denotes empty elements, e.g., plan bodies, contexts, intentions. Atoms (at) can be labelled with sources. Two kinds of goals are defined, achieve goals (!) and test goals (?).

The operational semantics [5] of the language, is given by a set of rules that define a transition system between configurations \( \langle ag, C, M, T, s \rangle \), where:

- \( ag \) is an agent program formed by a set of beliefs \( bs \) and plans \( ps \).
- An agent circumstance \( C \) is a tuple \( \langle I, E, A \rangle \), where: \( I \) is a set of intentions \( \{i, i', \ldots\} \), each \( i \in I \) is a stack of partially instantiated plans \( p \in ps \); \( E \) is a
Table 1. Grammar of AgentSpeak(L) [5]

| \(ag\) ::= \(bs\) \(ps\)                                      | \(at\) ::= \(P(t_1, \ldots, t_n)\) \((n \geq 0)\)                     |
| \(bs\) ::= \(b_1 \ldots b_n\) \((n \geq 0)\)                                | \(a\) ::= \(P(t_1, \ldots, t_n)[s_1, \ldots, s_m]\) \((n \geq 0, m \geq 0)\) |
| \(ps\) ::= \(p_1 \ldots p_n\) \((n \geq 1)\)                             | \(s\) ::= \(\text{percept} | \text{self} | \text{id}\)            |
| \(p\) ::= \(te : ct \leftarrow h\)                                      | \(a\) ::= \(A(t_1, \ldots, t_n)\) \((n \geq 0)\)                    |
| \(te\) ::= \(+\text{at} | -\text{at} | +\text{g} | -\text{g}\)       | \(g\) ::= \(\text{lat} | ?\text{at}\)                                |
| \(ct\) ::= \(ct_1 | \top\)                                             | \(u\) ::= \(+b | -b\)                                               |
| \(ct_1\) ::= \(\text{at} | -\text{at} \top \text{\top} \top\)       | \(h\) ::= \(h_1; \top \top \top\)                                  |
| \(h_1\) ::= \(a | g | u | h_1; h_1\)                                    |                     |

set of events \(\{(te, i), (te', i'), \ldots\}\), each \(te\) is a trigger event and each \(i\) is an intention (internal events) or the empty intention \(\top\) (external events); and \(A\) is a set of actions to be performed in the environment.

– \(M\) is a tuple \(\langle In, Out, SI \rangle\) working as a mailbox, where: \(In\) is the mailbox of the agent; \(Out\) is a list of messages to be delivered by the agent; \(SI\) is a register of suspended intentions (intentions that wait for an answer message).

– \(T\) is a tuple \(\langle R, Ap, i, e, R \rangle\) that registers temporary information as follows: \(R\) is the set of relevant plans for a given event; \(Ap\) is the set of applicable plans (the subset of applicable plans which contexts are believed true); \(i, e, R\), and \(\rho\) register the current intention, event and applicable plan along one cycle of execution.

– The label \(s\) indicates the current step in the reasoning cycle of the agent.

Figure 2 shows the interpreter for AgentSpeak(L) as a transition system. The operational semantics rules [5] define the transitions. Because of space limitations, table 2 shows only some these rules.

3 Methods

In this section, the methodology and tools used to design and implement the MAS in this work, are introduced. First, the Prometheus [30] methodology for designing MAS, and its associated software PDT, are introduced; then the interpreter of AgentSpeak(L), Jason [27, 6, 28, 1, 35] and the KDD tool, WEKA [36], are briefly described.

3.1 The Prometheus Methodology and its PDT Tool

Prometheus [30] is a methodology for developing intelligent agents and MAS. It covers all the phases of development of a system – specification, design, implementation and testing/debugging. Prometheus consists of three main phases:
---

**System Specification.** First, the goals and subgoals of the system are identified. Also, the actors (persons or roles that interact with the system) and their interactions with the system in form of perceptions and actions, are specified. Then, some scenarios are created for each actor. These scenarios show the operation of the system and consist of a series of steps including: goals, other scenarios, perceptions and actions. When the goals and scenarios are specified completely, the similar goals are grouped to form roles. Particular perceptions and actions are assigned to these roles. Finally, each step in the scenarios is assigned to their role and the data requirements for these scenarios are identified.

**Architectural Design.** Based on the definitions generated in the previous phase, it is possible now to determine what kind of agents will be included in the system, as well as the interactions that will take place among them. To achieve this, several mechanisms are proposed, such as data coupling diagrams and agent acquaintance diagrams. In addition, in this phase is created the system overview diagram, that shows the structure of the system.

**Detailed Design.** The internal details of each agent are designed and it is specified how the agents will carry out their jobs. Each agent is refined in terms of its capacities, internal events, plans, and data structures.

Additionally, the open software PDT [29] gives support to the Prometheus methodology. This tool provides: a graphic interface that allows to develop the definitions (diagrams) obtained through the Prometheus methodology and assists in the maintenance of a sound design because it provides verification between the diagrams; automatic spread of design elements when it is possible and appropriate; and assistance in the search for names.

---

*Fig. 2. The interpreter for AgentSpeak(L) as a transition system.*
(SelEv) \( S_E(C_E) = \langle te, i \rangle \)  
\( \text{s.t. } C_E' = C_E \setminus \{ \langle te, i \rangle \} \)

(Rel1) \( T_e = \langle te, \ell \rangle, RelPlans(ag_{ps}, te) \neq \{ \} \)  
\( \text{s.t. } T'_e = RelPlans(ag_{ps}, te) \)

(Rel2) \( RelPlans(ps, te) = \{ \} \)  
\( \text{s.t. } T'_e = RelPlans(ps, te) \)

(Appl) \( AppPlans(ag_{ps}, T_e) \neq \{ \} \)  
\( \text{s.t. } T'_e = AppPlans(ag_{ps}, T_e) \)

(SelAppl) \( S_O(T_e) = \langle p, \ell \rangle \)  
\( \text{s.t. } T_e = \langle p, \ell \rangle \)

(ExtEv) \( T_e = \langle te, \top \rangle, T_e = \langle p, \ell \rangle \)  
\( \text{s.t. } C_I' = C_I \cup \{ \langle p, \ell \rangle \} \)

(SelInt1) \( C_I \neq \{ \} \)  
\( \text{s.t. } T_I = \langle p, \ell \rangle \)

(SelInt2) \( C_I = \{ \} \)  
\( \text{s.t. } T_I = \langle p, \ell \rangle \)

(AchvGl) \( T_e = \langle \text{head} - \ell, \text{lat}, h \rangle \)  
\( \text{s.t. } C_E' = C_E \cup \{ \langle \text{lat}, T_e \rangle \} \)

\( C_I' = C_I \setminus \{ T_e \} \)

(ClrInt1) \( T_e = \langle \text{head} - \top \rangle \)  
\( \text{s.t. } C_I' = C_I \setminus \{ T_e \} \)

(ClrInt2) \( T_e = \langle \text{head} - \top \rangle \)  
\( \text{s.t. } C_I' = (C_I \setminus \{ T_e \}) \cup \{ k[(\text{head} - h)\theta] \} \)  
\( \text{if } i = k[(\text{head} - g)\theta] \)  
\( \text{and } g\theta = TrEv(head) \)

(ClrInt3) \( T_e \neq \langle \text{head} - \top \rangle \)  
\( \text{s.t. } C_I' = C_I \setminus \{ T_e \} \)

Table 2. Some rules of the operational semantics of AgentSpeak(L).

3.2 Jason

Jason is an interpreter for an extended version of the AgentSpeak(L) agent programming language (see section 2.2). This interpreter is written in Java and implements the operational semantics of AgentSpeak(L) [27, 6] and its extensions [28, 1, 35]. The integrated develop environment of Jason provides a graphical interface, that allows to edit the configuration file of a MAS and the code of the agents written in AgentSpeak(L). Through of this environment is possible to control the execution of the MAS and distribute the agents over a network in a simple way. Another tool that comes with this environment is a “mind inspector” that allows observing the internal state of the agents in run time. Jason also includes: speech acts based on KQML for communication between agents; annotations of the plans for using selection functions based on decision theory;
selection functions configurable in Java; and mechanisms of extension and use of legacy code, by means of the “internal actions” defined by the user.

3.3 WEKA

WEKA [36] is a tool implemented in Java to perform experiments in a KDD process. It provides methods for preprocessing data, e.g., replacing the missing values and performing discretization on data sets; data mining algorithms, e.g., ID3, C4.5, NB, and TAN; and pattern evaluation algorithms as the stratified cross-validation method. Other algorithms to carry out each one of the tasks of data mining, but we have focused in the methods mentioned above due to our research lines. In addition, WEKA has tools for visualizing and preprocessing data [14]. The input of all algorithms takes the form of a relational table, that can be read from a file or generated through of a database query. The file can be in csv format or arff format, which is the native format of WEKA.

4 Explorations

This section describes our explorations for the BDI MAS support for the KDD process, from the design to the implementation of the system. Later, some results on the use of the system are reported.

4.1 Design

Figure 3 shows the overview diagram for the MAS implemented for this paper. The Coordinator agent perceives the requests for learning from the users of the system as well as the databases associated to the requests; and their format. If a database is in xls or csv formats, she converts it into arff format; in any another case, she asks the user for a database in one of the mentioned formats. Then, the Coordinator asks the Preprocessing agent to review the database. If it is not nominal, the former tells the user that the target must be of this kind, given the nature of the learning algorithms used by the agents; otherwise, the Coordinator prints “Class is nominal” and sends a preprocessing require to the Preprocessing agent. This agent replaces the missing values (with the mode when the attributes are nominal and the mean when these ones are numerical) and discretizes the database (supervised and non-supervised). Once Preprocessing informs to Coordinator that the database has been preprocessed, the latter sends learning requests to the ID3, C4.5, NB and TAN agents. These ones learn their respective models for both kinds of discretization of data, and inform the Coordinator of their results. The Coordinator selects and reports the winning model given some criteria, e.g., the most accurate model, the fastest result obtained, etc.

To obtain the system overview diagram is necessary to identify the goals and subgoals of the system in a goal overview diagram (System Specification phase). For example, the figure 4 shows the goal and its subgoals for preprocessing the
database: one of them verifies what kind the class is, another one replaces the missing values and the last one discretizes this database.

After that, the different possible scenarios for the system were defined in a scenario diagram (System Specification phase). For example, the figure 5 shows the scenario that will take place when the database has to be discretized supervisely, e.g., when the process of discretization takes into account the value of the class to create the different categories of data. It is worth mentioning that the discretization is based on the Minimum Description Length method (MDL) [10].

The roles of the system were created once the goals and scenarios were defined. These roles are formed by clustering similar goals, perceptions and actions in a system role diagram (System Specification phase). For example, the figure 6 shows the “Preprocessing of the DB” role. Observing the scenario shown in figure 5, we can see that both steps of the scenario are associated with this role.

The data coupling diagram and the agent-role grouping diagram (Architectural Design phase) were useful to identify the types of the agents in the MAS. Figure 7 shows the first of these diagrams, with the roles of the system and the data identified in the scenarios. Figure 8 shows the agent-role grouping di-
agram, which shows the roles assigned for each agent. Thus, the Coordinator is responsible for managing the input/output of the system and the database, etc.

In the last phase of this process (Detailed Design phase), each agent is refined in terms of its capacities, internal events, plans and structures of data. For example, the figure 9 shows the Preprocessing agent overview diagram, including two plans: “Verify the class” and “Preprocess the DB”. The first plan is executed when a request to verify the target class is perceived, then it executes the action that verifies the class and finally it sends the type of data of this target class.
4.2 Implementation

As mentioned, the MAS was implemented in AgentSpeak(L)'s interpreter Jason v.1.0 (section 3.2). The development was carried out in a laptop with the following characteristics: Fedora Core 5 as operative system, Intel(R) Pentium(R) M 1.70 GHz processor, 1 Gb in RAM, 80 Gb in Hard Drive.

According to the design described in the previous section, the MAS consists of six agents: Coordinator, Preprocessing, ID3, C4.5, NB y TAN. The features of each one of these agents are shown in the table 3. The code in table 4 shows the initial beliefs and some plans of the Coordinator agent.

<table>
<thead>
<tr>
<th>Coordinator</th>
<th>Preprocessing</th>
<th>ID3, C4.5, NB and TAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manage I/O</td>
<td>Replace missing values</td>
<td>Learn data mining models</td>
</tr>
<tr>
<td></td>
<td>Manage the database</td>
<td>Discretize the database</td>
</tr>
</tbody>
</table>

Table 3. Features of the agents.
// Coordinator in project metaclassif.mas2j
// BELIEFS
start. // Start the execution of the MAS
file("./DATABASES/Iris.csv").

// PLANS
@pi
+start: true
  <- .print("Coordinator agent, who manages the MAS...");
  ?file(PathDB);
  weka.verifyFormatDB(PathDB,Format);
  .print("DB given in format ",Format);
  !pVerifyFormatDB(PathDB,Format).

// Plans take actions depending on what type the database is
@pVFBD1
+!pVerifyFormatDB(PathDB,Format): not (Format == ".xls" | Format == ".csv" | Format == ".arff")
  <- .print("DB must be given in format .xls, .csv or .arff").

@pVFBD2
+!pVerifyFormatDB(PathDB,Format): Format == ".xls" | Format == ".csv" | Format == ".arff"
  <- !pConvertDB(PathDB,Format);
  ?file(PathDBArff);
  .send(preprocessing,achieve,verifyClass(PathDBArff));
  .wait("+ typeClass(TypeClass)");
  !printTClass.

Table 4. A fragment of the code for the Coordinator agent

<table>
<thead>
<tr>
<th>Plan</th>
<th>Internal Actions</th>
<th>WEKA Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>replaceMissing</td>
<td>ArffLoader, ArffSaver</td>
<td>Instances, Instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Filter, ReplaceMissingValues</td>
</tr>
<tr>
<td>@ppBD</td>
<td>discretizeS</td>
<td>ArffLoader, ArffSaver</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Instances, Instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Filter, Discretize</td>
</tr>
<tr>
<td>discretizeNS</td>
<td>ArffSaver</td>
<td>ArffSaver, Instances</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Instance, Filter, Filter</td>
</tr>
</tbody>
</table>

Table 5. Preprocessing Agent.

The internal actions used to implement the plan library for each agent are built from WEKA classes. Table 5 illustrates this for the Preprocessing agent. For example, the @ppBD plan uses the replaceMissing, discretizeS, and discretizeNS internal actions, implemented using methods inherited from the ArffLoader, ArffSaver classes, etc., provided by WEKA.
5 Tests and Results

The MAS was tested with different databases from the repository at the University of California [2] to observe its viability offering support. Table 6 describes databases used in the tests. Performance, in terms of accuracy, was estimated using the Stratified 10-fold Cross-Validation, since it has been reported [24] as the best method to select one model from a set of them. Tables 7 and 8 show the percentage of correctly classified instances for each model learned by the agents, with the databases discretized supervised and non-supervised, respectively.

<table>
<thead>
<tr>
<th>DB’s</th>
<th>Instances</th>
<th>Attributes</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anneal</td>
<td>798</td>
<td>39</td>
<td>5</td>
</tr>
<tr>
<td>Balance_Scale</td>
<td>625</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Credit_S</td>
<td>690</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>Diabetes</td>
<td>768</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>Ecoli</td>
<td>336</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>Hypothyroid</td>
<td>3163</td>
<td>26</td>
<td>2</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>351</td>
<td>35</td>
<td>2</td>
</tr>
<tr>
<td>Iris</td>
<td>150</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Lymphography</td>
<td>148</td>
<td>19</td>
<td>4</td>
</tr>
<tr>
<td>Segment</td>
<td>2310</td>
<td>20</td>
<td>7</td>
</tr>
<tr>
<td>Soybean</td>
<td>307</td>
<td>36</td>
<td>19</td>
</tr>
<tr>
<td>Vehicle</td>
<td>846</td>
<td>19</td>
<td>4</td>
</tr>
<tr>
<td>Zoo</td>
<td>101</td>
<td>18</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 6. Description of the Databases.

The performance results show that there is no such a thing as the “best method” and the MAS automatizes the search for the best classifier using the selected algorithms. More importantly, the MAS avoids getting the “worst method” results, which in some cases (Zoo/ID3 and Ecoli/ID3) is quite relevant. The dynamics of the experiments suggests that the coordinator or another new agent, can be programmed to produce comparative reports, instead of the “the winner is...” behavior. Tables 9 and 10 provide the time taken to build the models learned by the agents, with the data discretized supervised and non-supervised, respectively.

The results of these tables indicate us that for both kinds of data discretization, NB always builds its models in the least time (because it only learns the parameters of the model, since this one is always the same) than the rest of the classifiers, whereas TAN is the one takes the most time to learn its models. In addition, we can see that when the data are discretized non-supervised, in general, the models are built in the least time.
Table 7. Percentage of correctly classified instances obtained with the data discretized supervised.

Table 8. Percentage of correctly classified instances obtained with the data discretized non-supervised.

6 Conclusions

This paper presents the design and implementation of a BDI MAS to support the KDD process. More precisely we present a framework that uses methodology and tools proposed in the MAS literature, to approach the automatizing of support for the use of common tools, as WEKA, in the KDD process. The MAS was designed according to the Prometheus methodology with the support of the
Table 9. Time taken to build the models learned with data discretized supervisely.

<table>
<thead>
<tr>
<th>DB’s</th>
<th>%ID3</th>
<th>%C4.5</th>
<th>%NB</th>
<th>%TAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anneal</td>
<td>0.96</td>
<td>0.91</td>
<td>0.22</td>
<td>2.48</td>
</tr>
<tr>
<td>Balance_Scale</td>
<td>0.24</td>
<td>0.41</td>
<td>0.11</td>
<td>0.16</td>
</tr>
<tr>
<td>Credit_S</td>
<td>0.6</td>
<td>0.78</td>
<td>0.15</td>
<td>0.53</td>
</tr>
<tr>
<td>Diabetes</td>
<td>0.66</td>
<td>0.81</td>
<td>0.2</td>
<td>0.42</td>
</tr>
<tr>
<td>Ecoli</td>
<td>1.62</td>
<td>1.28</td>
<td>0.3</td>
<td>1.65</td>
</tr>
<tr>
<td>Hypothyroid</td>
<td>1.57</td>
<td>1.15</td>
<td>0.33</td>
<td>3.25</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>0.63</td>
<td>0.65</td>
<td>0.21</td>
<td>1.47</td>
</tr>
<tr>
<td>Iris</td>
<td>0.31</td>
<td>0.17</td>
<td>0.1</td>
<td>0.26</td>
</tr>
<tr>
<td>Lymphography</td>
<td>0.48</td>
<td>0.46</td>
<td>0.21</td>
<td>0.37</td>
</tr>
<tr>
<td>Segment</td>
<td>1.04</td>
<td>0.57</td>
<td>0.16</td>
<td>2.72</td>
</tr>
<tr>
<td>Soybean</td>
<td>0.48</td>
<td>0.2</td>
<td>0.05</td>
<td>3.48</td>
</tr>
<tr>
<td>Vehicle</td>
<td>0.51</td>
<td>0.44</td>
<td>0.04</td>
<td>1.3</td>
</tr>
<tr>
<td>Zoo</td>
<td>0.12</td>
<td>0.05</td>
<td>0.01</td>
<td>0.53</td>
</tr>
<tr>
<td>μ</td>
<td>0.71</td>
<td>0.61</td>
<td>0.16</td>
<td>1.43</td>
</tr>
</tbody>
</table>

Table 10. Time taken to build the models learned with data discretized non-supervised.

<table>
<thead>
<tr>
<th>DB’s</th>
<th>%ID3</th>
<th>%C4.5</th>
<th>%NB</th>
<th>%TAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anneal</td>
<td>0.34</td>
<td>0.14</td>
<td>0.02</td>
<td>0.33</td>
</tr>
<tr>
<td>Balance_Scale</td>
<td>0.17</td>
<td>0.11</td>
<td>0</td>
<td>0.02</td>
</tr>
<tr>
<td>Credit_S</td>
<td>0.15</td>
<td>0.08</td>
<td>0.01</td>
<td>0.07</td>
</tr>
<tr>
<td>Diabetes</td>
<td>0.2</td>
<td>0.13</td>
<td>0</td>
<td>0.06</td>
</tr>
<tr>
<td>Ecoli</td>
<td>0.31</td>
<td>0.08</td>
<td>0</td>
<td>0.28</td>
</tr>
<tr>
<td>Hypothyroid</td>
<td>0.51</td>
<td>0.31</td>
<td>0.06</td>
<td>0.56</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>0.21</td>
<td>0.06</td>
<td>0.01</td>
<td>0.14</td>
</tr>
<tr>
<td>Iris</td>
<td>0.06</td>
<td>0.13</td>
<td>0</td>
<td>0.04</td>
</tr>
<tr>
<td>Lymphography</td>
<td>0.12</td>
<td>0.06</td>
<td>0</td>
<td>0.16</td>
</tr>
<tr>
<td>Segment</td>
<td>0.4</td>
<td>0.18</td>
<td>0.02</td>
<td>0.37</td>
</tr>
<tr>
<td>Soybean</td>
<td>0.14</td>
<td>0.13</td>
<td>0.01</td>
<td>2.73</td>
</tr>
<tr>
<td>Vehicle</td>
<td>0.34</td>
<td>0.11</td>
<td>0.02</td>
<td>0.28</td>
</tr>
<tr>
<td>Zoo</td>
<td>0.13</td>
<td>0.06</td>
<td>0.02</td>
<td>0.29</td>
</tr>
<tr>
<td>μ</td>
<td>0.24</td>
<td>0.12</td>
<td>0.01</td>
<td>0.41</td>
</tr>
</tbody>
</table>

PDT tool. Some advantages that were found when using Prometheus to model the KDD process like a BDI MAS are:

– The KDD process was clearly understood, because of the goal oriented analysis to define the MAS. The use of diagrams in the design was also helpful.
– The transition between the design and implementation of the BDI MAS was really easy. The diagrams obtained with Prometheus are expressed in terms of beliefs, goals, plans, events, etc., which are the natural constructors for the BDI agents and its AgentSpeak(L) programming language.
Prometheus can be used to model any process within an organization, since its diagrams offer a high level of abstraction, in the sense that they can be referred in a similar language to ours (for example, through beliefs, goals, scenarios, roles, plans, events, actions, etc.).

The MAS was implemented in the AgentSpeak(L) agent programming language, through the Jason interpreter. The main point in favor of Jason, regarding other development platforms of MAS (as JACK [7], JAM [21], JADEX [31], among others), is its theoretical basis, which gets to implement the operational semantics of AgentSpeak(L).

Java helps since both WEKA and Jason are written in this language, so that inherit methods in both senses was natural. We have opted for taking methods in WEKA to built internal actions in Jason agents. Future work will explore another possibility: considering WEKA as the environment for the MAS implemented in Jason. This should enable a richer interaction between the agents and the users of WEKA, e.g., an agent can execute directly commands from WEKA, in the same way the user does. Another contribution due to Java is portability. The MAS has been executed successfully on Windows, Linux, Solaris, Mac OS X, all for free.

Our exploration provides the basis to built more elaborated MAS in this context, e.g, extending the number of agents (more learning algorithms adopted) or extending their competences (wiser agents using better the algorithms). A much more ambitious goal is to approach meta-learning by this way, e.g., the agents learn intentionally [17, 18] to become wiser. This kind of learning enables the agents to learn when a given plan is really useful, given the desires and beliefs of an agent, and its past experience, i.e., agents that learn to learn. Given that, such a system would be much more elaborated, we are currently developing formal tools for AgentSpeak(L) program verification [19].

Acknowledgments. The second author is supported by the CONACYT scholarship 197800 and DIP-UJAT (DAIS-02 UJAT-EGRESADA/2005) agreement.

References